

## 7.2 Data Collection through API

CPE311 Computational Thinking with Python

**Name:** Quibral, Juliann Vincent B.

**Section:** CPE22S3

**Performed on:** 03/13/2024

**Submitted on:** 03/13/2024

**Submitted to:** Engr. Roman M. Richard

### ✓ Using the NCEI API

```

1 import requests
2
3 def make_request(endpoint, payload=None):
4
5     return requests.get(
6         f'https://www.ncdc.noaa.gov/cdo-web/api/v2/{endpoint}',
7         headers={
8             'token': 'ScaDZizCDQEmIzVblxYzxvHktsb1PcCK'
9         },
10        params=payload
11    )

1 # see what datasets are available
2 response = make_request('datasets', {'startdate': '2018-10-01'})
3 response.status_code
4
5     200

```

### ✓ Get the key of the result

```

1 response.json().keys()
2
3     dict_keys(['metadata', 'results'])

1 response.json()['metadata']
2
3     {'resultset': {'offset': 1, 'count': 11, 'limit': 25}}

1 response.json()['results'][0].keys()

```

```
1 response.json()['results'][0].keys()
dict_keys(['uid', 'mindate', 'maxdate', 'name', 'datacoverage', 'id'])
```

## ✓ parse the result

```
1 [(data['id'], data['name']) for data in response.json()['results']]
[('GHCND', 'Daily Summaries'),
 ('GSOM', 'Global Summary of the Month'),
 ('GSOY', 'Global Summary of the Year'),
 ('NEXRAD2', 'Weather Radar (Level II)'),
 ('NEXRAD3', 'Weather Radar (Level III)'),
 ('NORMAL_ANN', 'Normals Annual/Seasonal'),
 ('NORMAL_DLY', 'Normals Daily'),
 ('NORMAL_HLY', 'Normals Hourly'),
 ('NORMAL_MLY', 'Normals Monthly'),
 ('PRECIP_15', 'Precipitation 15 Minute'),
 ('PRECIP_HLY', 'Precipitation Hourly')]
```

## ✓ Figure which category we want

```
1 # get data category id
2 response = make_request(
3 'datacategories',
4 payload={
5 'datasetid' : 'GHCND'
6 }
7 )
8 response.status_code
200
```

```
1 response.json()['results']
[{'name': 'Evaporation', 'id': 'EVAP'},
 {'name': 'Land', 'id': 'LAND'},
 {'name': 'Precipitation', 'id': 'PRCP'},
 {'name': 'Sky cover & clouds', 'id': 'SKY'},
 {'name': 'Sunshine', 'id': 'SUN'},
 {'name': 'Air Temperature', 'id': 'TEMP'},
 {'name': 'Water', 'id': 'WATER'},
 {'name': 'Wind', 'id': 'WIND'},
 {'name': 'Weather Type', 'id': 'WXTYPE'}]
```

## ✓ Grab the data type ID for the Temperature category

```

1 # get data type id
2 response = make_request(
3 'datatypes',
4 payload={
5 'datacategoryid' : 'TEMP',
6 'limit' : 100
7 }
8 )
9 response.status_code
    200

1 [(datatype['id'], datatype['name']) for datatype in response.json()['results']][-5:]
    [
      ('MNTM', 'Monthly mean temperature'),
      ('TAVG', 'Average Temperature.'),
      ('TMAX', 'Maximum temperature'),
      ('TMIN', 'Minimum temperature'),
      ('TOBS', 'Temperature at the time of observation')]

1 [(datatype['id'], datatype['maxdate']) for datatype in response.json()['results']][-5]
    [
      ('MNTM', '2016-03-01'),
      ('TAVG', '2024-03-11'),
      ('TMAX', '2024-03-11'),
      ('TMIN', '2024-03-11'),
      ('TOBS', '2024-03-11')]

1 [(datatype['id'], datatype['mindate']) for datatype in response.json()['results']][-5]
    [
      ('MNTM', '1763-01-01'),
      ('TAVG', '1750-02-01'),
      ('TMAX', '1750-02-01'),
      ('TMIN', '1750-02-01'),
      ('TOBS', '1876-11-27')]

```

## ✓ Determine which Location Category we want

```

1 # get location category id
2 response = make_request(
3 'locationcategories',
4 {
5 'datasetid' : 'GHCND'
6 }
7 )
8 response.status_code
    200

```

```

1 import pprint
2 pprint.pprint(response.json())

{'metadata': {'resultset': {'count': 12, 'limit': 25, 'offset': 1}},
 'results': [{'id': 'CITY', 'name': 'City'},
              {'id': 'CLIM_DIV', 'name': 'Climate Division'},
              {'id': 'CLIM_REG', 'name': 'Climate Region'},
              {'id': 'CNTRY', 'name': 'Country'},
              {'id': 'CNTY', 'name': 'County'},
              {'id': 'HYD_ACC', 'name': 'Hydrologic Accounting Unit'},
              {'id': 'HYD_CAT', 'name': 'Hydrologic Cataloging Unit'},
              {'id': 'HYD_REG', 'name': 'Hydrologic Region'},
              {'id': 'HYD_SUB', 'name': 'Hydrologic Subregion'},
              {'id': 'ST', 'name': 'State'},
              {'id': 'US_TERR', 'name': 'US Territory'},
              {'id': 'ZIP', 'name': 'Zip Code'}]}

```

## ✓ Get NYC Location ID

```

1 def get_item(name, what, endpoint, start=1, end=None):
2
3     # find the midpoint which we use to cut the data in half each time
4     mid = (start + (end if end else 1)) // 2
5     # lowercase the name so this is not case-sensitive
6     name = name.lower()
7     # define the payload we will send with each request
8     payload = {
9         'datasetid' : 'GHCND',
10        'sortfield' : 'name',
11        'offset' : mid, # we will change the offset each time
12        'limit' : 1 # we only want one value back
13    }
14    # make our request adding any additional filter parameters from `what`
15    response = make_request(endpoint, **payload, **what)
16
17    if response.ok:
18        # if response is ok, grab the end index from the response metadata the first time
19        end = end if end else response.json()['metadata']['resultset']['count']
20        # grab the lowercase version of the current name
21        current_name = response.json()['results'][0]['name'].lower()
22        # if what we are searching for is in the current name, we have found our item
23        if name in current_name:
24            return response.json()['results'][0] # return the found item
25        else:
26            if start >= end:
27                # if our start index is greater than or equal to our end, we couldn't find it
28                return {}
29            elif name < current_name:

```

```

29     elif name < current_name:
30         # our name comes before the current name in the alphabet, so we search further
31         return get_item(name, what, endpoint, start, mid - 1)
32     elif name > current_name:
33         # our name comes after the current name in the alphabet, so we search further to
34         return get_item(name, what, endpoint, mid + 1, end)
35     else:
36         # response wasn't ok, use code to determine why
37         print(f'Response not OK, status: {response.status_code}')
38
39 def get_location(name):
40
41     return get_item (name, {'locationcategoryid' : 'CITY'}, 'locations')

1 nyc = get_location('New York')
2 nyc

{'mindate': '1869-01-01',
 'maxdate': '2024-03-11',
 'name': 'New York, NY US',
 'datacoverage': 1,
 'id': 'CITY:US360019'}

1 Jry = get_location('Jersey ')
2 Jry

{'mindate': '1869-01-01',
 'maxdate': '2024-03-11',
 'name': 'Jersey City, NJ US',
 'datacoverage': 1,
 'id': 'CITY:US340001'}

```

## ✓ Get the station ID for Central Park

```

1 central_park = get_item('NY City Central Park', {'locationid' : nyc['id']}, 'stations')
2 central_park

{'elevation': 42.7,
 'mindate': '1869-01-01',
 'maxdate': '2024-03-10',
 'latitude': 40.77898,
 'name': 'NY CITY CENTRAL PARK, NY US',
 'datacoverage': 1,
 'id': 'GHCND:USW00094728',
 'elevationUnit': 'METERS',
 'longitude': -73.96925}

```

## ✓ Get the station ID for Statue of Liberty

```
1 Statue_of_Liberty = get_item('NY City Statue_of_Liberty', {'locationid' : nyc['id']},
2 Statue_of_Liberty
    Response not OK, status: 502
```

## ✓ Request the temperature data

```
1 # get NYC daily summaries data
2 response = make_request(
3 'data',
4 {
5 'datasetid' : 'GHCND',
6 'stationid' : central_park['id'],
7 'locationid' : nyc['id'],
8 'startdate' : '2018-10-01',
9 'enddate' : '2018-10-31',
10 'datatypeid' : ['TMIN', 'TMAX', 'TOBS'], # temperature at time of observation, min, ar
11 'units' : 'metric',
12 'limit' : 1000
13 }
14 )
15 response.status_code
    200
```

## ✓ Create a DataFrame

```
1 import pandas as pd
2 df = pd.DataFrame(response.json()['results'])
3 df.head()
```

---

Next steps: [View recommended plots](#)

```

1 df.datatype.unique()

array(['TMAX', 'TMIN'], dtype=object)

1 if get_item(
2 'NY City Central Park', {'locationid' : nyc['id'], 'datatypeid': 'TOBS'}, 'stations'
3 ):
4     print('Found!')

Found!

```

## ✓ Using a different station

```

1 laguardia = get_item(
2 'LaGuardia', {'locationid' : nyc['id']}, 'stations'
3 )
4 laguardia

{'elevation': 3,
 'mindate': '1939-10-07',
 'maxdate': '2024-03-11',
 'latitude': 40.77945,
 'name': 'LAGUARDIA AIRPORT, NY US',
 'datacoverage': 1,
 'id': 'GHCND:USW00014732',
 'elevationUnit': 'METERS',
 'longitude': -73.88027}

1 # get NYC daily summaries data
2 response = make_request(
3 'data',
4 {
5 'datasetid' : 'GHCND',
6 'stationid' : laguardia['id'],
7 'locationid' : nyc['id'],
8 'startdate' : '2018-10-01',
9 'enddate' : '2018-10-31',
10 'datatypeid' : ['TMIN', 'TMAX', 'TAVG'], # temperature at time of observation, min, ar
11 'units' : 'metric',
12 'limit' : 1000
13 }
14 )
15 response.status_code

200

```

```
1 f = pd.DataFrame(response.json()['results'])
2 df.head()
```

---

Next steps:

 [View recommended plots](#)

```
1 df.datatype.value_counts()
      TMAX      31
      TMIN      31
      Name: datatype, dtype: int64

1 df.to_csv('sample_data/nyc_temperatures.csv', index=False)
```

✓ *\*End of 7.2 \**

```
1 file_path = "/content/nyc_temperatures.csv"
```



